ROB521: Assignment 3

Drini Kerciku — 1004750780

April 6, 2023

1 Occupancy Map Algorithm

The mapped environment represents the ground truth from Assignment 2 and improves on the layout of the labyrinth and objects located in it. A balance between the factors α and β needs to found when populating the map in order to obtain reasonably uniform contour of the wall - we are converting Cartesian coordinates to discrete grid coordinates and rounding error will cause cavities to appear along the walls, past the walls or within the walls (more noticeable at door passages).

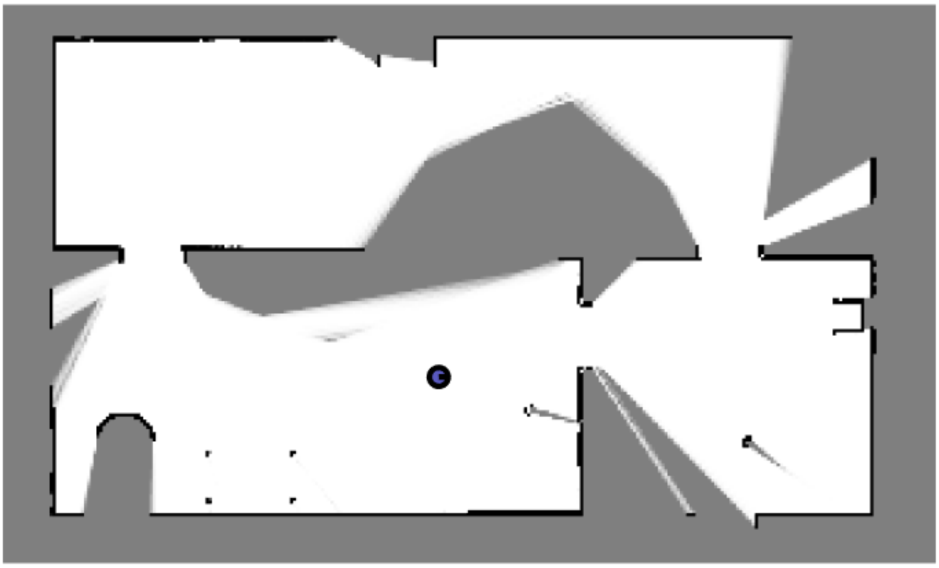

Figure 1: Mapping of the occupancy grid.

2 Particle Filter Localization

The results do not match the reference output exactly as we can observe a spike in the particle filter error when entering room three, which can be accounted to large input data ($\omega > 0.1\ rad/s$) in that particular sector coupled with missing data from the original map on the right side of the room. However, the particle filter manages to recover quickly and maintain lower error than the dead-reckoning approach - another possible fix for this would be in selecting a different distance metric when updating the weights, or more robust computations in the expected laser reading.

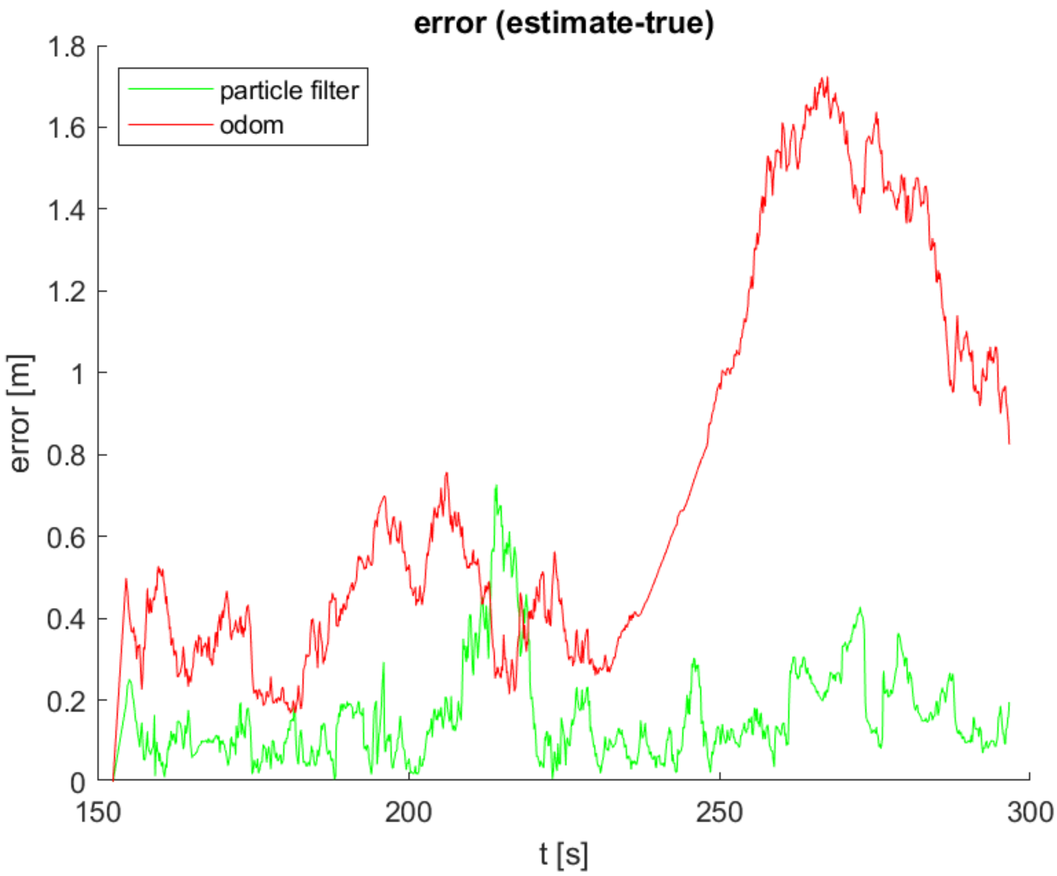

Figure 2: Dead-reckoning v. particle filter localization.

3 Source Code

3.1 Part I: Occupancy Mapping Algorithm

```
% =========
% ass3_q1.m
% =========
%
% This assignment will introduce you to the idea of first building an
% occupancy grid then using that grid to estimate a robot's motion using a
% particle filter.
%
% There are two questions to complete (5 marks each):
%
%    Question 1: code occupancy mapping algorithm
%    Question 2: see ass3_q2.m
%
% Fill in the required sections of this script with your code, run it to
% generate the requested plot/movie, then paste the plots into a short report
% that includes a few comments about what you've observed.  Append your
% version of this script to the report.  Hand in the report as a PDF file
% and the two resulting AVI files from Questions 1 and 2.
%
% requires: basic Matlab, 'gazebo.mat'
%
% T D Barfoot, January 2016
%
clear all; close all; clc;

% set random seed for repeatability
rng(1);

% ==========================
% load the dataset from file
% ==========================
%
%    ground truth poses: t_true x_true y_true theta_true
% odometry measurements: t_odom v_odom omega_odom
%           laser scans: t_laser y_laser
%    laser range limits: r_min_laser r_max_laser
%    laser angle limits: phi_min_laser phi_max_laser
%
load gazebo.mat;

% =======================================
% Question 1: build an occupancy grid map
% =======================================
%
% Write an occupancy grid mapping algorithm that builds the map from the
% perfect ground-truth localization.  Some of the setup is done for you
% below.  The resulting map should look like "ass2_q1_soln.png".  You can
% watch the movie "ass2_q1_soln.mp4" to see what the entire mapping process
% should look like.  At the end you will save your occupancy grid map to
% the file "occmap.mat" for use in Question 2 of this assignment.

% allocate a big 2D array for the occupancy grid
ogres = 0.05;                   % resolution of occ grid
ogxmin = -7;                    % minimum x value
ogxmax = 8;                     % maximum x value
ogymin = -3;                    % minimum y value
ogymax = 6;                     % maximum y value
ognx = (ogxmax-ogxmin)/ogres;   % number of cells in x direction
ogny = (ogymax-ogymin)/ogres;   % number of cells in y direction
oglo = zeros(ogny,ognx);        % occupancy grid in log-odds format
ogp = zeros(ogny,ognx);         % occupancy grid in probability format

% precalculate some quantities
numodom = size(t_odom,1);
npoints = size(y_laser,2);
angles = linspace(phi_min_laser, phi_max_laser,npoints);
dx = ogres*cos(angles);
dy = ogres*sin(angles);

```

```
% interpolate the noise-free ground-truth at the laser timestamps
t_interp = linspace(t_true(1),t_true(numodom),numodom);
x_interp = interp1(t_interp,x_true,t_laser);
y_interp = interp1(t_interp,y_true,t_laser);
theta_interp = interp1(t_interp,theta_true,t_laser);
omega_interp = interp1(t_interp,omega_odom,t_laser);

% set up the plotting/movie recording
vid = VideoWriter('ass2_q1.avi');
open(vid);
figure(1);
clf;
pcolor(ogp);
colormap(1-gray);
shading('flat');
axis equal;
axis off;
M = getframe;
writeVideo(vid,M);

% precalculate some quantities
cos_angles = cos(angles);
sin_angles = sin(angles);
offset = [0.1; 0];
map0 = [ogxmin; ogymin];

alpha = 2.5;
beta = 0.15;

% loop over laser scans (every fifth)
for i=1:5:size(t_laser,1)

    % ------insert your occupancy grid mapping algorithm here------

    if abs(omega_interp(i)) < 0.1

        % extract robot pose in 2D space
        thetaR = theta_interp(i);
        ptR = [x_interp(i); y_interp(i)];
        R_i = [cos(-thetaR), sin(-thetaR);
               -sin(-thetaR), cos(-thetaR)];

        % transfrom all laser scans to robot's local frame in one go
        RLaser_i = y_laser(i, :);
        QLaser_i = [RLaser_i.*cos_angles; RLaser_i.*sin_angles];

        QGlobal = R_i*(QLaser_i - offset) + ptR - map0;

        % parse entries 1-by-1 to update the log likelihoods
        for j = 1 : npoints

            % skip NaN data entries
            if isnan(RLaser_i(j))
                continue
            end

            % check ranges to accumulate log likely hoods
            for r = r_min_laser : ogres : r_max_laser

                if r < RLaser_i(j)

                    % transform range in global frame
                    rG = ptR + R_i*(r*[cos_angles(j); sin_angles(j)] - offset) - map0;

                    % reward empty region
                    oglo(floor(rG(2)/ogres), floor(rG(1)/ogres)) = oglo(floor(rG(2)/ogres), floor(rG(1)/ogres)) - beta;

                end

            end

            % penalize occupied region
```

```
            oglo(floor(QGlobal(2,j)/ogres), floor(QGlobal(1,j)/ogres)) = oglo(floor(
    QGlobal(2, j)/ogres), floor(QGlobal(1,j)/ogres)) + alpha;

        end
    end

    % convert log probabilities to probabilities
    for ogpX = 1:size(ogp, 1)
        for ogpY = 1:size(ogp, 2)
            ogp(ogpX, ogpY) = exp(oglo(ogpX, ogpY)) / (1 + exp(oglo(ogpX, ogpY)));
        end
    end

    % ------end of your occupancy grid mapping algorithm-------

    % draw the map
    clf;
    pcolor(ogp);
    colormap(1-gray);
    shading('flat');
    axis equal;
    axis off;

    % draw the robot
    hold on;
    x = (x_interp(i)-ogxmin)/ogres;
    y = (y_interp(i)-ogymin)/ogres;
    th = theta_interp(i);
    r = 0.15/ogres;
    set(rectangle( 'Position', [x-r y-r 2*r 2*r], 'Curvature', [1 1]),'LineWidth',2,'
    FaceColor',[0.35 0.35 0.75]);
    set(plot([x x+r*cos(th)]', [y y+r*sin(th)]', 'k-'),'LineWidth',2);

    % save the video frame
    M = getframe;
    writeVideo(vid,M);

    pause(0.1);

end

close(vid);
print -dpng ass2_q1.png

save occmap.mat ogres ogxmin ogxmax ogymin ogymax ognx ogny oglo ogp;
```

3.2 Part II: Particle Filter Algorithm

```
% =========
% ass3_q2.m
% =========
%
% This assignment will introduce you to the idea of first building an
% occupancy grid then using that grid to estimate a robot's motion using a
% particle filter.
%
% There are three questions to complete (5 marks each):
%
%    Question 1: see ass3_q1.m
%    Question 2: code particle filter to localize from known map
%
% Fill in the required sections of this script with your code, run it to
% generate the requested plot/movie, then paste the plots into a short report
% that includes a few comments about what you've observed.  Append your
% version of this script to the report.  Hand in the report as a PDF file
% and the two resulting AVI files from Questions 1 and 2.
%
% requires: basic Matlab, 'gazebo.mat', 'occmap.mat'
%
% T D Barfoot, January 2016
%
clear all; close all; clc;

```

```
% set random seed for repeatability
rng(1);

% ==========================
% load the dataset from file
% ==========================
%
%    ground truth poses: t_true x_true y_true theta_true
% odometry measurements: t_odom v_odom omega_odom
%           laser scans: t_laser y_laser
%    laser range limits: r_min_laser r_max_laser
%    laser angle limits: phi_min_laser phi_max_laser
%
load gazebo.mat;

% ================================================
% load the occupancy map from question 1 from file
% ================================================
%  ogres: resolution of occ grid
% ogxmin: minimum x value
% ogxmax: maximum x value
% ogymin: minimum y value
% ogymax: maximum y value
%   ognx: number of cells in x direction
%   ogny: number of cells in y direction
%   oglo: occupancy grid in log-odds format
%    ogp: occupancy grid in probability format
load occmap.mat;

% =========================================================================
% Question 2: localization from an occupancy grid map using particle filter
% =========================================================================
%
% Write a particle filter localization algorithm to localize from the laser
% rangefinder readings, wheel odometry, and the occupancy grid map you
% built in Question 1.  We will only use two laser scan lines at the
% extreme left and right of the field of view, to demonstrate that the
% algorithm does not need a lot of information to localize fairly well.  To
% make the problem harder, the below lines add noise to the wheel odometry
% and to the laser scans.  You can watch the movie "ass2_q2_soln.mp4" to
% see what the results should look like.  The plot "ass2_q2_soln.png" shows
% the errors in the estimates produced by wheel odometry alone and by the
% particle filter look like as compared to ground truth; we can see that
% the errors are much lower when we use the particle filter.

% interpolate the noise-free ground-truth at the laser timestamps
numodom = size(t_odom,1);
t_interp = linspace(t_true(1),t_true(numodom),numodom);
x_interp = interp1(t_interp,x_true,t_laser);
y_interp = interp1(t_interp,y_true,t_laser);
theta_interp = interp1(t_interp,theta_true,t_laser);
omega_interp = interp1(t_interp,omega_odom,t_laser);

% interpolate the wheel odometry at the laser timestamps and
% add noise to measurements (yes, on purpose to see effect)
v_interp = interp1(t_interp,v_odom,t_laser) + 0.2*randn(size(t_laser,1),1);
omega_interp = interp1(t_interp,omega_odom,t_laser) + 0.04*randn(size(t_laser,1),1);

% add noise to the laser range measurements (yes, on purpose to see effect)
% and precompute some quantities useful to the laser
y_laser = y_laser + 0.1*randn(size(y_laser));
npoints = size(y_laser,2);
angles = linspace(phi_min_laser, phi_max_laser,npoints);
dx = ogres*cos(angles);
dy = ogres*sin(angles);
y_laser_max = 5;   % don't use laser measurements beyond this distance

% particle filter tuning parameters (yours may be different)
nparticles = 200;       % number of particles
v_noise = 0.2;          % noise on longitudinal speed for propagating particle
u_noise = 0.2;          % noise on lateral speed for propagating particle
omega_noise = 0.04;     % noise on rotational speed for propagating particle
laser_var = 0.5^2;      % variance on laser range distribution
```

```
w_gain = 10*sqrt( 2 * pi * laser_var );     % gain on particle weight

% generate an initial cloud of particles
x_particle = x_true(1) + 0.5*randn(nparticles,1);
y_particle = y_true(1) + 0.3*randn(nparticles,1);
theta_particle = theta_true(1) + 0.1*randn(nparticles,1);

% compute a wheel odometry only estimate for comparison to particle
% filter
x_odom_only = x_true(1);
y_odom_only = y_true(1);
theta_odom_only = theta_true(1);

% error variables for final error plots - set the errors to zero at the start
pf_err(1) = 0;
wo_err(1) = 0;

% set up the plotting/movie recording
vid = VideoWriter('ass2_q2.avi');
open(vid);
figure(2);
clf;
hold on;
pcolor(ogp);
set(plot( (x_particle-ogxmin)/ogres, (y_particle-ogymin)/ogres, 'g.' ),'MarkerSize',10,'Color',[0 0.6 0]);
set(plot( (x_odom_only-ogxmin)/ogres, (y_odom_only-ogymin)/ogres, 'r.' ),'MarkerSize',20);
x = (x_interp(1)-ogxmin)/ogres;
y = (y_interp(1)-ogymin)/ogres;
th = theta_interp(1);
r = 0.15/ogres;
set(rectangle( 'Position', [x-r y-r 2*r 2*r], 'Curvature', [1 1]),'LineWidth',2,'FaceColor',[0.35 0.35 0.75]);
set(plot([x x+r*cos(th)]', [y y+r*sin(th)]', 'k-'),'LineWidth',2);
set(plot( (mean(x_particle)-ogxmin)/ogres, (mean(y_particle)-ogymin)/ogres, 'g.' ),'MarkerSize',20);
colormap(1-gray);
shading('flat');
axis equal;
axis off;
M = getframe;
writeVideo(vid,M);

% loop over laser scans
for i=2:size(t_laser,1)

    % update the wheel-odometry-only algorithm
    dt = t_laser(i) - t_laser(i-1);
    v = v_interp(i);
    omega = omega_interp(i);
    x_odom_only = x_odom_only + dt*v*cos( theta_odom_only );
    y_odom_only = y_odom_only + dt*v*sin( theta_odom_only );
    phi = theta_odom_only + dt*omega;
    while phi > pi
        phi = phi - 2*pi;
    end
    while phi < -pi
        phi = phi + 2*pi;
    end
    theta_odom_only = phi;

    % loop over the particles
    for n=1:nparticles

        % propagate the particle forward in time using wheel odometry
        % (remember to add some unique noise to each particle so they
        % spread out over time)
        v = v_interp(i) + v_noise*randn(1);
        u = u_noise*randn(1);
        omega = omega_interp(i) + omega_noise*randn(1);
        x_particle(n) = x_particle(n) + dt*(v*cos( theta_particle(n) ) - u*sin( theta_particle(n) ));
```

```
        y_particle(n) = y_particle(n) + dt*(v*sin( theta_particle(n) ) + u*cos(
    theta_particle(n) ));
        phi = theta_particle(n) + dt*omega;
        while phi > pi
            phi = phi - 2*pi;
        end
        while phi < -pi
            phi = phi + 2*pi;
        end
        theta_particle(n) = phi;

        % pose of particle in initial frame
        T = [cos(theta_particle(n)) -sin(theta_particle(n)) x_particle(n); ...
             sin(theta_particle(n))  cos(theta_particle(n)) y_particle(n); ...
                     0                      0                  1];

        % compute the weight for each particle using only 2 laser rays
        % (right=beam 1 and left=beam 640)
        w_particle(n) = 1.0;
        for beam=1:2

            % we will only use the first and last laser ray for
            % localization
            if beam==1  % rightmost beam
                j = 1;
            elseif beam==2  % leftmost beam
                j = 640;
            end

            % -----insert your particle filter weight calculation here -----

            % compute bearing angle in robot's frame of reference
            phi_j_particle = angles(j);

            % initialize varibles to 0
            xGrid = 0;
            yGrid = 0;
            R = 0;

            % iterate through viable ranges towards the bearing angle
            % direction
            for r = r_min_laser : ogres : r_max_laser

                % pose transformation to global coordinates and grid (x,y)
                rParticle = r*[cos(phi_j_particle); sin(phi_j_particle)] - [0.1; 0];
                rGlobal = T*[rParticle; 1];

                xGrid = round((rGlobal(1) - ogxmin)/ogres);
                yGrid = round((rGlobal(2) - ogymin)/ogres);

                % adjust values
                if xGrid > ognx
                    xGrid = ognx;
                end

                if yGrid > ogny
                    yGrid = ogny;
                end

                if yGrid < 1
                    yGrid = 1;
                end

                if xGrid < 1
                    xGrid = 1;
                end

                % check the value of the occupancy map
                if ogp(yGrid, xGrid) > 0.5
                    R = r;
                    break
                end

```

```
            end

            % compute gain
            w_particle(n) = w_gain*w_particle(n)*normpdf(y_laser(i,j),...
                    R, sqrt(laser_var));

            % ------end of your particle filter weight calculation-------
        end

    end

    % resample the particles using Madow systematic resampling
    w_bounds = cumsum(w_particle)/sum(w_particle);
    w_target = rand(1);
    j = 1;
    for n=1:nparticles
       while w_bounds(j) < w_target
           j = mod(j,nparticles) + 1;
       end
       x_particle_new(n) = x_particle(j);
       y_particle_new(n) = y_particle(j);
       theta_particle_new(n) = theta_particle(j);
       w_target = w_target + 1/nparticles;
       if w_target > 1
           w_target = w_target - 1.0;
           j = 1;
       end
    end
    x_particle = x_particle_new;
    y_particle = y_particle_new;
    theta_particle = theta_particle_new;

    % save the translational error for later plotting
    pf_err(i) = sqrt( (mean(x_particle) - x_interp(i))^2 + (mean(y_particle) - y_interp(i))^2 );
    wo_err(i) = sqrt( (x_odom_only - x_interp(i))^2 + (y_odom_only - y_interp(i))^2 );

    % plotting
    figure(2);
    clf;
    hold on;
    pcolor(ogp);
    set(plot( (x_particle-ogxmin)/ogres, (y_particle-ogymin)/ogres, 'g.' ),'MarkerSize',10,'Color',[0 0.6 0]);
    set(plot( (x_odom_only-ogxmin)/ogres, (y_odom_only-ogymin)/ogres, 'r.' ),'MarkerSize',20);
    x = (x_interp(i)-ogxmin)/ogres;
    y = (y_interp(i)-ogymin)/ogres;
    th = theta_interp(i);
    if ~isnan(y_laser(i,1)) & y_laser(i,1) <= y_laser_max
       set(plot([x x+y_laser(i,1)/ogres*cos(th+angles(1))]', [y y+y_laser(i,1)/ogres*sin(th+angles(1))]', 'm-'),'LineWidth',1);
    end
    if ~isnan(y_laser(i,640)) & y_laser(i,640) <= y_laser_max
       set(plot([x x+y_laser(i,640)/ogres*cos(th+angles(640))]', [y y+y_laser(i,640)/ogres*sin(th+angles(640))]', 'm-'),'LineWidth',1);
    end
    r = 0.15/ogres;
    set(rectangle( 'Position', [x-r y-r 2*r 2*r], 'Curvature', [1 1]),'LineWidth',2,'FaceColor',[0.35 0.35 0.75]);
    set(plot([x x+r*cos(th)]', [y y+r*sin(th)]', 'k-'),'LineWidth',2);
    set(plot( (mean(x_particle)-ogxmin)/ogres, (mean(y_particle)-ogymin)/ogres, 'g.' ),'MarkerSize',20);
    colormap(1-gray);
    shading('flat');
    axis equal;
    axis off;

    % save the video frame
    M = getframe;
    writeVideo(vid,M);

    pause(0.005);
```

```

end

close(vid);

% final error plots
figure(3);
clf;
hold on;
plot( t_laser, pf_err, 'g-' );
plot( t_laser, wo_err, 'r-' );
xlabel('t [s]');
ylabel('error [m]');
legend('particle filter', 'odom', 'Location', 'NorthWest');
title('error (estimate-true)');
print -dpng ass2_q2.png
```